ENSF 338 Lab 3

Group 35

Exercise 1

1. Implement a basic version of that function, using the code seen in class for the rest of the algorithm [0.2 pts]

Code:

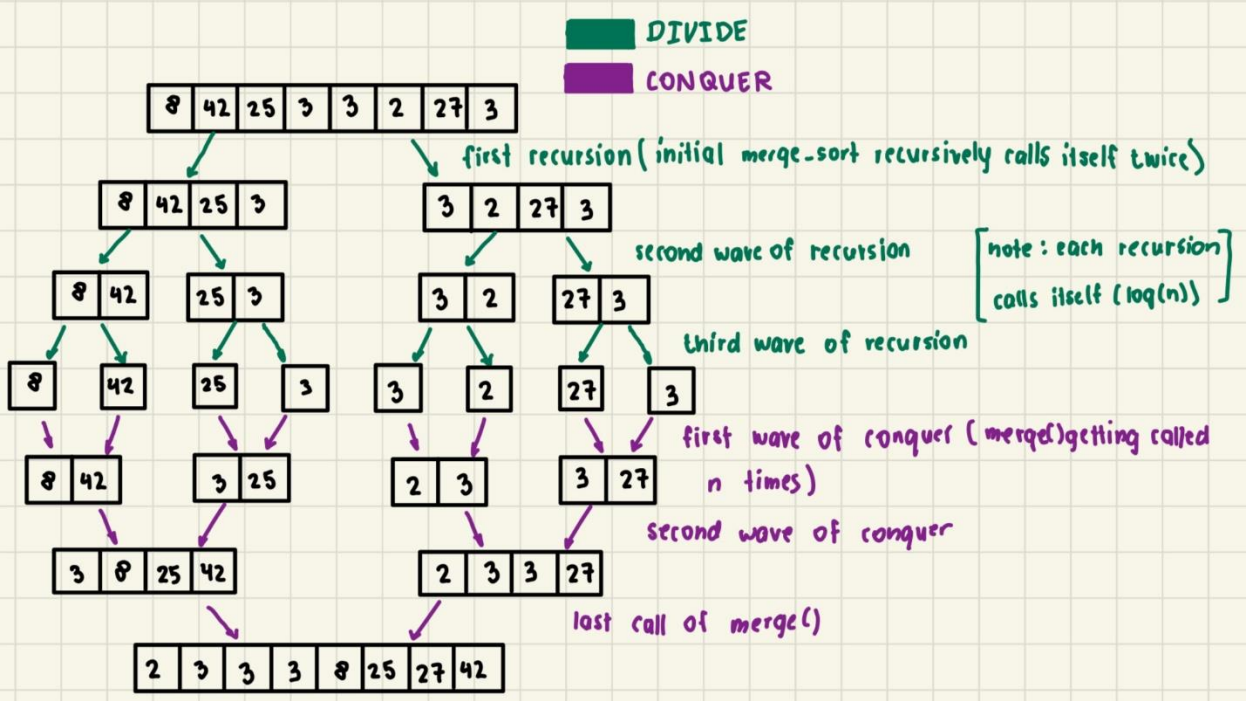
```
def merge(arr, low, mid, high):
    n1 = mid - low + 1; n2 = high - mid     # gets the length of two arrays
    L = [0] * (n1); R = [0] * (n2)          # Made Two Temp Arrays
    p = 0; g = 0; k = low       # intial index values for sub arrays, and merged
subarray

    # TWO For Loop that copies Data into the two temporary arrays
    for i in range(0, n1):
        L[i] = arr[low + i]

    for j in range(0, n2):
        R[j] = arr[mid + 1 + j]

    while p < n1 and g < n2:
        if L[p] <= R[g]:
            arr[k] = L[p]
            p += 1
        else:
            arr[k] = R[g]
            g += 1
        k += 1

    # Dealing with remaining elements of both temporary arrays
    while p < n1:
        arr[k] = L[p]
        p += 1
        k += 1

    while g < n2:
        arr[k] = R[g]
        g += 1
        k += 1

def merge_sort(arr, low, high):
    mid = (low + high) // 2
```

```
    print(mid)
    if low < high:
        merge_sort(arr, low, mid)
        merge_sort(arr, mid+1, high)
    merge(arr, low, mid, high)


# Driver code to test above
arr = [8, 42, 25, 3, 3, 2, 27, 3]
print("\nInitial array is")
for i in range(len(arr)):
    print("%d" % arr[i],end=" ")

merge_sort(arr, 0, len(arr)-1)
print("\n\nSorted array is")
for i in range(len(arr)):
    print("%d" % arr[i],end=" ")
```

2. Argue that the overall algorithm has a worst-case complexity of O(nlogn). Note, your description must specifically refer to the code you wrote, i.e., not just generically talk about mergesort. [0.4 pts].

The initial "divide" function has a complexity of O(n) because it calls itself twice recursively per call and the "conquer" function is called n number of times. That is why no matter which case we refer to, the overall algorithm as a complexity of O(nlogn).

3. Manually apply your algorithm to the input below, showing each step (similar to the example seen in class) until the algorithm completes and the vector is fully sorted. Explanation should include both visuals (vector at each step) and discussion [0.4 pts]



4. Is the number of steps consistent with your complexity analysis? Justify your answer. [0.2 pts]

Yes, because our "divide" function splits of into two branches, making it $O(\log(n))$, then the "conquer" function joins it back again, making it $O(n)$. Together the total complexity is $O(n\log(n))$.